orthog — Orthogonalize variables and compute orthogonal polynomials

Description

orthog orthogonalizes a set of variables, creating a new set of orthogonal variables (all of type double), using a modified Gram–Schmidt procedure (Golub and Van Loan 2013). The order of the variables determines the orthogonalization; hence, the “most important” variables should be listed first.

Execution time is proportional to the square of the number of variables. With many (>10) variables, orthog will be fairly slow.

orthopoly computes orthogonal polynomials for one variable.

Quick start

Generate ox1, ox2, and ox3 containing orthogonalized versions of x1, x2, and x3

orthog x1 x2 x3, generate(ox1 ox2 ox3)

Same as above

orthog x1 x2 x3, generate(ox*)

Generate op1, op2, and op3 containing degree 1, 2, and 3 orthogonal polynomials for x1

orthopoly x1, generate(op1 op2 op3) degree(3)

Same as above

orthopoly x1, generate(op1-op3) degree(3)

As above, and generate matrix op containing coefficients of the orthogonal polynomials

orthopoly x1, generate(op1-op3) degree(3) poly(op)

Menu

orthog

Data > Create or change data > Other variable-creation commands > Orthogonalize variables

orthopoly

Data > Create or change data > Other variable-creation commands > Orthogonal polynomials
Orthogonalize variables

```plaintext
orthog [varlist] [if] [in] [weight], generate(newvarlist) [matrix(matname)]
```

Compute orthogonal polynomial

```plaintext
orthpoly varname [if] [in] [weight],
{generate(newvarlist) | poly(matname)} [degree(#)]
```

orthpoly requires that `generate(newvarlist)` or `poly(matname)`, or both, be specified.
`varlist` may contain time-series operators; see [U] 11.4.4 Time-series varlists.
`iweights`, `aweights`, `fweights`, and `pweights` are allowed, see [U] 11.1.6 weight.

Options for orthog

- `generate(newvarlist)` is required. `generate()` creates new orthogonal variables of type double.
  - For orthog, `newvarlist` will contain the orthogonalized `varlist`. If `varlist` contains `d` variables, then so will `newvarlist`. `newvarlist` can be specified by giving a list of exactly `d` new variable names, or it can be abbreviated using the styles `newvar1-newvar` or `newvar*`. For these two styles of abbreviation, new variables `newvar1`, `newvar2`, ..., `newvar` are generated.

- `matrix(matname)` creates a `(d+1) x (d+1)` matrix containing the matrix `R` defined by `X = QR`, where `X` is the `N x (d+1)` matrix representation of `varlist` plus a column of ones and `Q` is the `N x (d+1)` matrix representation of `newvarlist` plus a column of ones (`d` = number of variables in `varlist`, and `N` = number of observations).

Options for orthopoly

- `generate(newvarlist)` or `poly()`, or both, must be specified. `generate()` creates new orthogonal variables of type double. `newvarlist` will contain orthogonal polynomials of degree 1, 2, ..., `d` evaluated at `varname`, where `d` is as specified by `degree(d)`. `newvarlist` can be specified by giving a list of exactly `d` new variable names, or it can be abbreviated using the styles `newvar1-newvar` or `newvar*`. For these two styles of abbreviation, new variables `newvar1`, `newvar2`, ..., `newvar` are generated.

- `poly(matname)` creates a `(d+1) x (d+1)` matrix called `matname` containing the coefficients of the orthogonal polynomials. The orthogonal polynomial of degree `i ≤ d` is

  ```plaintext
  matname[i, d+1] + matname[i, 1]*varname + matname[i, 2]*varname^2 + \cdots + matname[i, i]*varname^i
  ```

  The coefficients corresponding to the constant term are placed in the last column of the matrix. The last row of the matrix is all zeros, except for the last column, which corresponds to the constant term.

- `degree(#)` specifies the highest-degree polynomial to include. Orthogonal polynomials of degree 1, 2, ..., `d = #` are computed. The default is `d = 1`. 

Remarks and examples

Orthogonal variables are useful for two reasons. The first is numerical accuracy for highly collinear variables. Stata’s `regress` and other estimation commands can face much collinearity and still produce accurate results. But, at some point, these commands will drop variables because of collinearity. If you know with certainty that the variables are not perfectly collinear, you may want to retain all their effects in the model. If you use `orthog` or `orthpoly` to produce a set of orthogonal variables, all variables will be present in the estimation results.

Users are more likely to find orthogonal variables useful for the second reason: ease of interpreting results. `orthog` and `orthpoly` create a set of variables such that the “effects” of all the preceding variables have been removed from each variable. For example, if we issue the command

```
. orthog x1 x2 x3, generate(q1 q2 q3)
```

the effect of the constant is removed from `x1` to produce `q1`; the constant and `x1` are removed from `x2` to produce `q2`; and finally the constant, `x1`, and `x2` are removed from `x3` to produce `q3`. Hence,

\[
q_1 = r_{01} + r_{11} x_1 \\
q_2 = r_{02} + r_{12} x_1 + r_{22} x_2 \\
q_3 = r_{03} + r_{13} x_1 + r_{23} x_2 + r_{33} x_3
\]

This effect can be generalized and written in matrix notation as

\[
X = QR
\]

where \(X\) is the \(N \times (d + 1)\) matrix representation of `varlist` plus a column of ones, and \(Q\) is the \(N \times (d + 1)\) matrix representation of `newvarlist` plus a column of ones (\(d\) = number of variables in `varlist` and \(N\) = number of observations). The \((d + 1) \times (d + 1)\) matrix \(R\) is a permuted upper-triangular matrix, that is, \(R\) would be upper triangular if the constant were first, but the constant is last, so the first row/column has been permuted with the last row/column. Because Stata’s estimation commands list the constant term last, this allows \(R\), obtained via the `matrix()` option, to be used to transform estimation results.

Example 1: orthog

Consider Stata’s `auto.dta` dataset. Suppose that we postulate a model in which `price` depends on the car’s `length`, `weight`, `headroom`, and trunk size (`trunk`). These predictors are collinear, but not extremely so—the correlations are not that close to 1:

```
. use https://www.stata-press.com/data/r16/auto
   (1978 Automobile Data)
. correlate length weight headroom trunk
   (obs=74)
```

<table>
<thead>
<tr>
<th></th>
<th>length</th>
<th>weight</th>
<th>headroom</th>
<th>trunk</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>1.0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>0.9460</td>
<td>1.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>headroom</td>
<td>0.5163</td>
<td>0.4835</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>trunk</td>
<td>0.7266</td>
<td>0.6722</td>
<td>0.6620</td>
<td>1.0000</td>
</tr>
</tbody>
</table>
regress certainly has no trouble fitting this model:

```
. regress price length weight headroom trunk
```

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>df</th>
<th>MS</th>
<th>Number of obs = 74</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>236016580</td>
<td>4</td>
<td>59004145</td>
<td>F(4, 69) = 10.20</td>
</tr>
<tr>
<td>Residual</td>
<td>399048816</td>
<td>69</td>
<td>5783316.17</td>
<td>Prob &gt; F = 0.0000</td>
</tr>
<tr>
<td>Total</td>
<td>635065396</td>
<td>73</td>
<td>8699525.97</td>
<td>Adj R-squared = 0.3352</td>
</tr>
</tbody>
</table>

| price | Coef.    | Std. Err. | t     | P>|t| | [95% Conf. Interval] |
|-------|----------|-----------|-------|------|----------------------|
| length | -101.7092 | 42.12534 | -2.41 | 0.018 | -185.747   -17.67147 |
| weight | 4.753066  | 1.120054 | 4.24  | 0.000 | 2.518619   6.987512 |
| headroom | -711.5679 | 445.0204 | -1.60 | 0.114 | -1599.359  176.2256 |
| trunk | 114.0859  | 109.9488 | 1.04  | 0.303 | -105.2559  333.4277 |
| _cons | 11488.47  | 4543.902 | 2.53  | 0.014 | 2423.638   20553.31 |

However, we may believe a priori that length is the most important predictor, followed by weight, headroom, and trunk. We would like to remove the “effect” of length from all the other predictors, remove weight from headroom and trunk, and remove headroom from trunk. We can do this by running orthog, and then we fit the model again using the orthogonal variables:

```
. orthog length weight headroom trunk, gen(olength oweight oheadroom otrunk)
> matrix(R)

. regress price olength oweight oheadroom otrunk
```

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>df</th>
<th>MS</th>
<th>Number of obs = 74</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>236016580</td>
<td>4</td>
<td>59004145</td>
<td>F(4, 69) = 10.20</td>
</tr>
<tr>
<td>Residual</td>
<td>399048816</td>
<td>69</td>
<td>5783316.17</td>
<td>R-squared = 0.3716</td>
</tr>
<tr>
<td>Total</td>
<td>635065396</td>
<td>73</td>
<td>8699525.97</td>
<td>Adj R-squared = 0.3352</td>
</tr>
</tbody>
</table>

| price | Coef.    | Std. Err. | t     | P>|t| | [95% Conf. Interval] |
|-------|----------|-----------|-------|------|----------------------|
| olength | 1265.049  | 279.5584 | 4.53  | 0.000 | 707.3454   1822.753 |
| oweight | 1175.765  | 279.5584 | 4.21  | 0.000 | 687.4617   1733.469 |
| oheadroom | -349.9916 | 279.5584 | -1.25 | 0.215 | -907.6955  207.7122 |
| otrunk | 290.0776  | 279.5584 | 1.04  | 0.303 | -267.6262  847.7815 |
| _cons | 6165.257  | 279.5584 | 22.05 | 0.000 | 5607.553   6722.961 |

Using the matrix R, we can transform the results obtained using the orthogonal predictors back to the metric of original predictors:

```
. matrix b = e(b)*inv(R)'
. matrix list b
```

```
               b[1,5]
length   weight   headroom   trunk   _cons
y1         -101.70924  4.7530659  -711.56789  114.08591  11488.475
```

**Technical note**

The matrix $R$ obtained using the `matrix()` option with orthog can also be used to recover $X$ (the original `varlist`) from $Q$ (the orthogonalized `newvarlist`), one variable at a time. Continuing with the previous example, we illustrate how to recover the trunk variable:
orthog — Orthogonalize variables and compute orthogonal polynomials

matrix C = R[1...,"trunk"]
matrix score double rtrunk = C
compare rtrunk trunk
count minimum average maximum
difference
rtrunk>trunk 74 8.88e-15 1.91e-14 3.55e-14
jointly defined 74 8.88e-15 1.91e-14 3.55e-14
total 74

Here the recovered variable rtrunk is almost exactly the same as the original trunk variable. When you are orthogonalizing many variables, this procedure can be performed to check the numerical soundness of the orthogonalization. Because of the ordering of the orthogonalization procedure, the last variable and the variables near the end of the varlist are the most important ones to check.

The orthopoly command effectively does for polynomial terms what the orthog command does for an arbitrary set of variables.

Example 2: orthopoly

Again consider the auto.dta dataset. Suppose that we wish to fit the model

\[
\text{mpg} = \beta_0 + \beta_1 \text{weight} + \beta_2 \text{weight}^2 + \beta_3 \text{weight}^3 + \beta_4 \text{weight}^4 + \epsilon
\]

We will first compute the regression with natural polynomials:

\[
\text{. generate double } w1 = \text{weight}
\]
\[
\text{. generate double } w2 = w1*w1
\]
\[
\text{. generate double } w3 = w2*w1
\]
\[
\text{. generate double } w4 = w3*w1
\]
\[
\text{. correlate } w1-w4
\]
\[
\text{(obs=74)}
\]
\[
\begin{array}{c|cccc}
\text{w1} & \text{w2} & \text{w3} & \text{w4} \\
\hline
\text{w1} & 1.0000 & & & \\
\text{w2} & 0.9915 & 1.0000 & & \\
\text{w3} & 0.9665 & 0.9916 & 1.0000 & \\
\text{w4} & 0.9279 & 0.9679 & 0.9922 & 1.0000 \\
\end{array}
\]
\[
\text{. regress } \text{mpg} \text{ w1-w4}
\]

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>df</th>
<th>MS</th>
<th>Number of obs = 74</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>1652.73666</td>
<td>4</td>
<td>413.184164</td>
<td>F(4, 69) = 36.06</td>
</tr>
<tr>
<td>Residual</td>
<td>790.722803</td>
<td>69</td>
<td>11.4597508</td>
<td>Prob &gt; F = 0.0000</td>
</tr>
<tr>
<td>Total</td>
<td>2443.45946</td>
<td>73</td>
<td>33.4720474</td>
<td>R-squared = 0.6764</td>
</tr>
</tbody>
</table>

| mp0 | Coef. | Std. Err. | t | P>|t| | [95% Conf. Interval] |
|-----|-------|-----------|---|------|---------------------|
| w1  | .0289302 | .1161939 | 0.25 | 0.804 | -.2028704 .2607307 |
| w2  | -.0000229 | .0000566 | -0.40 | 0.687 | -.0001359 .0000901 |
| w3  | 5.74e-09 | 1.19e-08 | 0.48 | 0.631 | -.180e-08 2.95e-08 |
| w4  | -4.86e-13 | 9.14e-13 | -0.53 | 0.596 | -2.31e-12 1.34e-12 |
| _cons | 23.94421 | 86.60667 | 0.28 | 0.783 | -148.8314 196.7198 |
Some of the correlations among the powers of `weight` are very large, but this does not create any problems for `regress` However, we may wish to look at the quadratic trend with the constant removed, the cubic trend with the quadratic and constant removed, etc. `orthpoly` will generate polynomial terms with this property:

\[
\begin{align*}
\text{. orthpoly weight, generate(pw*) deg(4) poly(P)} \\
\text{. regress mpg pw1-pw4}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>df</th>
<th>MS</th>
<th>Number of obs = 74</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>1652.73666</td>
<td>4</td>
<td>413.184164</td>
<td>F(4, 69) = 36.06</td>
</tr>
<tr>
<td>Residual</td>
<td>790.722803</td>
<td>69</td>
<td>11.4597508</td>
<td>Prob &gt; F = 0.0000</td>
</tr>
<tr>
<td>Total</td>
<td>2443.45946</td>
<td>73</td>
<td>33.4720474</td>
<td>R-squared = 0.6764</td>
</tr>
</tbody>
</table>

| mpg     | Coef.   | Std. Err. | t   | P>|t| | [95% Conf. Interval] |
|---------|---------|-----------|-----|------|---------------------|
| pw1     | -4.638252 | .3935245  | -11.79 | 0.000 | -5.423312 to -3.853192 |
| pw2     | .8263545  | .3935245  | 2.10  | 0.039 | .0412947 to 1.611414  |
| pw3     | -.3068616 | .3935245  | -0.78 | 0.438 | -1.091921 to .4781982 |
| pw4     | -.209457  | .3935245  | -0.53 | 0.596 | -.9945168 to .5756028 |
| _cons   | 21.2973   | .3935245  | 54.12 | 0.000 | 20.51224 to 22.08236  |

Compare the \( p \)-values of the terms in the natural polynomial regression with those in the orthogonal polynomial regression. With orthogonal polynomials, it is easy to see that the pure cubic and quartic trends are not significant and that the constant, linear, and quadratic terms each have \( p < 0.05 \).

The matrix \( P \) obtained with the `poly()` option can be used to transform coefficients for orthogonal polynomials to coefficients for natural polynomials:

\[
\begin{align*}
\text{. orthpoly weight, poly(P) deg(4)} \\
\text{. matrix b = e(b)*P} \\
\text{. matrix list b} \\
\text{b[1,5]} \\
\text{deg1 deg2 deg3 deg4 _cons} \\
\text{y1 .02893016 -.00002291 5.745e-09 -4.862e-13 23.944212}
\end{align*}
\]

**Methods and formulas**

`orthog`’s orthogonalization can be written in matrix notation as

\[
X = QR
\]

where \( X \) is the \( N \times (d + 1) \) matrix representation of `varlist` plus a column of ones and \( Q \) is the \( N \times (d + 1) \) matrix representation of `newvarlist` plus a column of ones (\( d = \) number of variables in `varlist`, and \( N = \) number of observations). The \( (d + 1) \times (d + 1) \) matrix \( R \) is a permuted upper-triangular matrix; that is, \( R \) would be upper triangular if the constant were first, but the constant is last, so the first row/column has been permuted with the last row/column.

\( Q \) and \( R \) are obtained using a modified Gram–Schmidt procedure; see Golub and Van Loan (2013, 254–255) for details. The traditional Gram–Schmidt procedure is notoriously unsound, but the modified procedure is good. `orthog` performs two passes of this procedure.

`orthopoly` uses the Christoffel–Darboux recurrence formula (Abramowitz and Stegun 1964).
Both `orthog` and `orthpoly` normalize the orthogonal variables such that

\[ Q'WQ = MI \]

where \( W = \text{diag}(w_1, w_2, \ldots, w_N) \) with weights \( w_1, w_2, \ldots, w_N \) (all 1 if weights are not specified), and \( M \) is the sum of the weights (the number of observations if weights are not specified).

References


Also see

[R] `regress` — Linear regression